Acta Scientific COMPUTER SCIENCES

Volume 6 Issue 5 May 2024

Review Article

Autonomous Mobile Robot Obstacle Avoidance with Reinforcement Learning

Shravan S Rai*

Arizona State University, USA

***Corresponding Author:** Shravan S Rai, Arizona State University, USA.

Received: April 17, 2024

Published: April 30, 2024

Abstract

This paper presents an innovative approach to obstacle avoidance in autonomous systems through the application of reinforcement learning techniques. This comprehensive study delves into the intricate challenges of navigating autonomous systems in dynamic environments. By harnessing the power of advanced reinforcement learning techniques, specifically a synergistic blend of Q-learning and the Deep Deterministic Policy Gradient (DDPG) algorithm, this effort has pioneered a groundbreaking approach to improve obstacle avoidance capabilities of a mobile robot. This research meticulously explores the implementation and efficacy of this model through rigorous testing in a variety of simulated scenarios. The results obtained are not only promising but signify a substantial leap forward from traditional methods. There are marked enhancements observed in both the efficiency and safety aspects of autonomous navigation, paving the way for more sophisticated and resilient obstacle avoidance strategies. This investigation not only makes a significant contribution to the evolving field of robotics and artificial intelligence but also lays the groundwork for future explorations into the potential applications of reinforcement learning in complex, ever-changing environments. The findings in this paper offer valuable insights and a solid foundation for subsequent research aimed at optimizing autonomous systems for enhanced operational performance and adaptability.

Keywords: Obstacle; Avoidance; Reinforcement; Learning; DDPG

Introduction

In the realm of autonomous robotics, mastering the art of navigation in unknown environments is paramount. Traditional navigation methods, typically comprising localization, map building, and path planning, falter in uncharted territories due to their reliance on preexisting obstacle maps. These conventional approaches often struggle with complexity and are prone to getting trapped in local optima, especially in dynamic settings where collision avoidance is crucial. To address these limitations, heuristic methods, mirroring humanlike behavioral learning, are emerging as a promising alternative. This project introduces a novel approach using Deep Deterministic Policy Gradient (DDPG) Reinforcement Learning for navigating a mobile robot. The goal here is to enable the robot to travel from a starting point to a designated target location without colliding with static obstacles. The focus was on developing a simplified, yet universal model, leveraging information from the robot's immediate surroundings. This model optimizes the robot's path to the destination, enhancing its practical robustness and capability. Utilizing eight angular regions and six linear scales for directionality, the presented methodology enables the robot to select from one of eight angular directions at each time step, moving a fixed distance each time. This approach presents a universal characteristic, potentially applicable across diverse robotic platforms. In the simulation and training, the robot was liberated from the constraints of limited information, being allowed to access absolute angles and distances between itself and the target. However, during the test phase, the robot operates without this knowledge. The presented study delves into the application of reinforcement learning to guide a mobile robot through a static-obstacle-laden environment, modeled as a Markov Decision Process. This environment features distinct states and corresponding rewards, with the robot's objective being to maximize reward accumulation while avoiding obstacles. The robot's observed states include its own position, the goal position, linear and angular velocities, and proximity to obstacles. Its actions are incentivized through a reward system: small negative rewards for movement to encourage progress, significant negative rewards for collisions, and substantial positive rewards for reaching the goal. Additionally, incorporating a reward component for velocity aims to enhance the robot's speed. This project, thus, not only contributes to the field of autonomous robotics but also offers significant insights into the practical application of reinforcement learning for efficient and safe navigation in complex, obstacle-rich environments.

Related work

In the burgeoning field of autonomous vehicle technology, ensuring safety while navigating through obstacle laden environments is of paramount importance, necessitating advanced solutions for trajectory planning and real-time collision avoidance. Traditional strategies, while robust, often suffer from high computational demands. For instance, path planning with model-free controllers, such as those detailed by Grisetti, Grzonka, and Burgard [1] offers a computationally light approach but tends to be overly conservative. On the other hand, model-based methods like Model Predictive Control [2], balance slow global planning with rapid local avoidance but can be intensive in calculation and limited in scope, often requiring dynamic-specific models and precise estimation of obstacle positions from sensors. This project seeks to address these limitations by introducing a novel learning algorithm, which generates control commands directly from sensor inputs, thereby eliminating the need for obstacle position estimation. This approach, which is schematically summarized in 3 leverages neural networks for greater efficiency compared to traditional methods and integrates learning with motion planning. The most general approach for learning a control policy in such scenarios is model-free reinforcement learning, a class of methods that learns the control policy through direct interaction with the environment, as exemplified in the work of Silver., *et al*. [3] and others. This project builds upon these foundational concepts to develop an innovative solution for autonomous vehicles, combining the efficiency of direct sensor input-based control with the adaptability and learning capabilities of model-free reinforcement learning.

Methods

Q-Learning

Q-learning is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances. It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations. For any finite Markov decision process (FMDP), Q-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. Q-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy. ”Q” names the function that returns the reward used to provide the reinforcement and can be said to stand for the ”quality” of an action taken in a given state [11]. The weight for a step from a state δ_t steps into the future is calculated as $\gamma^{\delta t}$, where the discount factor which lies between 0 and 1 has the effect of valuing rewards received earlier higher than those received later thus, reflecting the value of a ”good start”).γ may also be interpreted as the probability to succeed at every δt. Thus, the algorithm can be approximated as shown in eq(1) [11].

$Q: S \times A \rightarrow R$ ----- (1)

The core of the algorithm is a Bellman equation as a simple value iteration update, using the weighted average of the old value and the new information as depicted in the below equation [11].

$$Q_{new}(s_t,a_t) \leftarrow Q_{old}(s_t,a_t) + \alpha(r_t + \gamma \max_a Q_{optimal}(s_{t+1},a) - Q_{old}(s_t,a_t)) \quad (2)$$

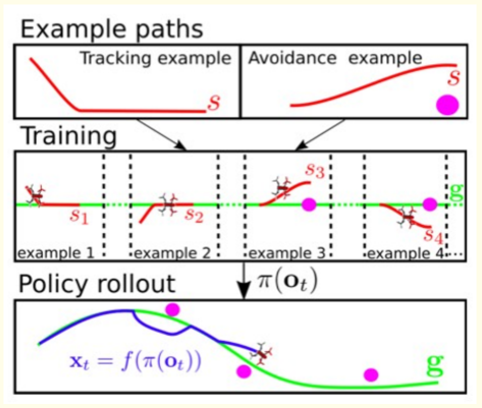

Figure 1: A policy is learned form few, short local collision avoidance and path following maneuvers (red). The learned policy generalizes to unseen scenes and can track long guidance paths (green) through complex environments while successfully avoiding obstacles (blue).

An episode of the algorithm ends when state s_{t+1} is a final or terminal state. However, Q-learning can also learn in non-episodic tasks. Since Q-learning is an iterative algorithm, it implicitly assumes an initial condition before the first update occurs. High initial values, also known as “optimistic initial conditions”, can encourage exploration, no matter what action is selected, the update rule will cause it to have lower values than the other alternative, thus increasing their choice probability. The first reward r can be used to reset the initial conditions [11].

Although for problems with discrete action space and smaller number of states, the Q learning algorithm works very well. However, for problems like navigation, balancing an inverted pendulum on a cart which have continuous action space, the Q learning algorithm is not feasible. This is because in continuous spaces finding the greedy policy requires an optimization of actions a_t at every time-step t. This optimization is too slow to be practical with large, unconstrained function approximators and nontrivial action spaces.

Deep Q-Learning

Q-learning is a simple yet quite powerful algorithm to create a cheat sheet for the agent. This helps the agent figure out exactly which action to perform [8]. It is pretty clear that one can’t infer the Q-value of new states from already explored states. This presents two problems:

- First, the amount of memory required to save and update that table would increase as the number of states increases
- Second, the amount of time required to explore each state to create the required Q-table would be unrealistic

In deep Q-learning, a neural network is used to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output [8]. The algorithm includes the following steps in sequence:

- All the past experience is stored by the user in memory
- The next action is determined by the maximum output of the Q-network
- The loss function here is mean squared error of the predicted Q-value and the target Q-value – Q*. This is basically a regression problem. However, we do not know the target or actual value here as we are dealing with a reinforcement learning problem.

Going back to the Q-value update equation derived from the Bellman equation. We have:

$$(r_{t+1} + \gamma \max Q_{optimal}(s_{t+1}, a)\ _a \qquad \text{-----}\ (2)$$

Eq(3) represents the target. One can argue that it is predicting its own value, but since R is the unbiased true reward, the network is going to update its gradient using back propagation to finally converge. Deep Q-Learning is depicted in the below figure.

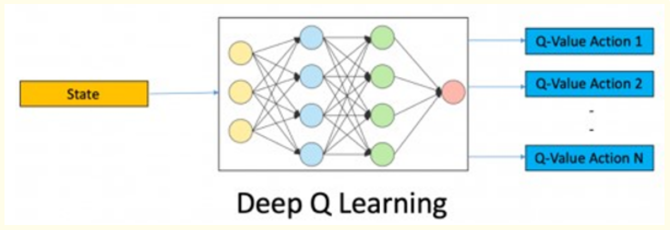

Figure 2: Pictorial Representation of Deep Q-Learning.

Deep deterministic policy gradient

Out of the numerous reinforcement learning algorithms available, Deep Deterministic Policy Gradient (DDPG) is best suited for continuous action space tasks. DDPG concurrently learns a Q-function and a policy μ. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy. This approach is closely connected to Q-learning, and is motivated the same way: if one knows the optimal action-value function $Q^*(s,a)$, then in any given state, the optimal action $a^*(s)$ can be found by solving.

$$a^*(s) = \underset{a}{\operatorname{argmax}} Q^*(s,a). \qquad (4)$$

DDPG interleaves learning an approximator to $Q^*(s,a)$ with learning an approximator to $a^*(s)$, and it does so in a way which is specifically adapted for environments with continuous action spaces. When there are a finite number of discrete actions, the max poses no problem, because the Q-values can be computed for each action separately and directly compare them. But when the action space is continuous, one can't exhaustively evaluate the space, and solving the optimization problem would make calculating $\max_a Q^*(s,a)$ a painfully expensive subroutine.

Because the action space is continuous, the function $Q^*(s,a)$ is presumed to be differentiable with respect to the action argument. This allows us to set up an efficient, gradient-based learning rule for a policy μ(s) which exploits that fact. Then, instead of running an expensive optimization subroutine each time we wish to compute $\max_a Q(s,a)$, we can approximate it with $\max_a Q(s,a) \approx Q(s,\mu(s))$.

DDPG consists of two types of Networks called the actor and the critic network. The actor network takes the environment states as its input and predicts the actions the agent will perform in the next time step, whereas the critic network is used for evaluating the policy function estimated by the actor according to the temporal difference (TD) error. This combination of actor network and critic network along with experience replay constitutes Deep Deterministic Policy Gradient algorithm (DDPG).

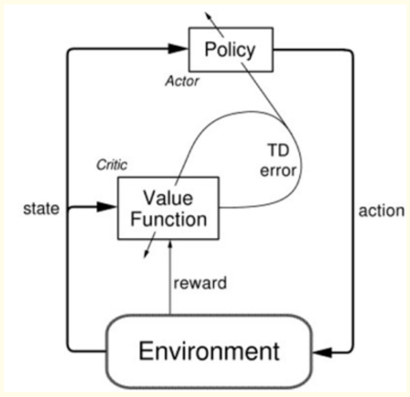

Figure 3: DDPG Architecture.

From [6], it is clear that the DDPG algorithm maintains a parameterized actor function $\mu(s \mid \theta^{\mu})$ which specifies the current policy by deterministically mapping states to a specific action. The critic Q(s,a) is learned using the Bellman equation as in Q-learning. The actor is updated by following the applying the chain rule to the expected return from the start distribution J with respect to the actor parameters [6]:

$$\nabla_{\theta^\mu} J = E_{st,\rho\beta}[\nabla_a Q(s,a|\theta^Q)_{s=s_t, a=\mu s_t} \times \qquad (5)$$
$$\nabla_{\theta^\mu} \mu(s|\theta_\mu)_{s=s_t}]$$

Deep deterministic policy gradient with hindsight experience replay

[4] Experience replay was first introduced by Lin (1992). The key idea of experience replay is to train the agent with the transitions sampled from the buffer of previously experienced transitions. A transition is defined to be a quadruple (s,a,r,s0), where s is the state, a is the action, r is the received reward after executing the action a in the state s and s0 is the next state. At each time step, the current transition is added to the replay buffer and some transitions are sampled from the replay buffer to train the agent.

[7] HER is an experience replay method which can be used to overcome the learning difficulties caused by the use of sparse rewards and avoid complex reward projects. Different from the traditional Reinforcement Learning methods, HER is proposed with a new parameter goal which consists of desired goal and achieved goal. The desired goal represents the task that the agent should accomplish. The achieved goal represents the task that the agent has completed at the current time. The key idea of HER is as follows: at some moment, although the agent has not achieved the desired goal, it has completed achieved goal. At this time, the desired goal can be replaced by the achieved goal so that the method can transform failed experiences into successful experiences and learn from them. In HER, even though the desired goal is not completed at present, if the learning process is repeated, the agent will complete the desired goal in the final so as to complete the task with the sparse rewards.

[9] Suppose the agent performs an episode of trying to reach goal state G from initial state S, but fails to do so and ends up in some state S' at the end of the episode. The trajectory is cached into our replay buffer:

$$(S_0,G,a_0,r_0,S_1),(S_1,G,a_1,r_1,S_2),\ldots,(S_n,G,a_n,r_n,S^0) \quad (6)$$

Where r with subscript k is the reward received at step k of the episode, and a with subscript k is the action taken at step k of the episode. The idea in HER is to imagine that the goal has actually been S0 all along, and that in this alternative reality the agent has reached the goal successfully and got the positive reward for doing so. So, in addition to caching the real trajectory as seen before, I also cache the following trajectory.

$$(S_0,S^0,a_0,r_0,S_1),(S_1,S^0,a_1,r_1,S_2),\ldots,(S_n,S^0,a_n,r_n,S^0) \quad (7)$$

This trajectory is the imagined one, and is motivated by the human ability to learn useful things from failed attempts. It should also be noted that in the imagined trajectory, the reward received at the final step of the episode is now a positive reward gained from reaching the imagined goal. By introducing the imagined trajectories to the replay buffer, it is ensured that no matter how bad the policy is, it will always have some positive rewards to learn from.

Implementation

In this project, a Simulation tool named CoppeliaSim (previously known as v-rep) by Coppelia Robotics is used along with OpenAI's gym environment [10]. The CoppeliaSim provides mobile robot and the scene in which the robot moves around. This when coupled with gym environment gives the states and information about the environment without any additional sensors. The environment looks like the.

The acquisition of environmental states and information is facilitated through the integration of PyRep [5], a comprehensive toolkit designed specifically for robot learning research. Built atop

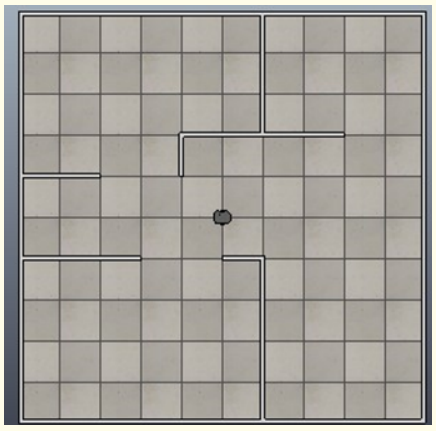
Figure 4: Simulation Environment.

CoppeliaSim, PyRep provides an effective interface with the OpenAI Gym environment, thereby enabling a seamless integration for conducting extensive reinforcement learning experiments.

In the development of the neural network architectures for both the actor and the critic components of the model, TensorFlow has been selected as the primary framework. This choice is predicated on TensorFlow's user friendly interface, coupled with its robust set of training functionalities that significantly enhance the efficiency of model training processes. The model architecture comprises four distinct networks: the actor, the critic, and their respective target networks – target actor and target critic. Each network is characterized by a shallow neural network architecture, incorporating two hidden layers with 200 and 100 neurons, respectively. The rectified linear unit (ReLU) activation function is employed across these layers due to its proven efficacy in facilitating the training process. Training of both the actor and the critic networks is executed utilizing the Adam optimizer, with the aim to minimize the mean square error between the outputs of the critic and the target critic networks.

The formulation of an effective reward function is paramount for the successful training of the agent. To this end, an all-encompassing reward function has been devised that accounts for the critical variables influencing the robot's performance. This function has been meticulously designed to encapsulate all pertinent aspects, with the reward equation delineated in Equation (8).

$$R = \begin{cases} 1, & d < d_{th} \\ -1, & \exists D < d_{collision} \\ -V_L 0.\cos(1,\theta), & \exists otherwise D < d_{proxth} \end{cases} \quad (8)$$

Where, V_L is the linear velocity of the mobile robot, θ is the heading angle, D is the vector distance read from the ultrasonic sensor, d is the distance between robot and goal position, $d_{collision}$ distance when collision occurs, d_{th} threshold distance between robot and goal position, d_{proxth} is the safety distance threshold for regular operation.

Results

The utilization of the CoppeliaSim robotic simulator facilitated the implementation of robot navigation within a two-dimensional environment. This simulation environment employed an array of ultrasonic sensors mounted on the virtual robot to detect obstacles in its vicinity. The collective detection range of these sensors is depicted in Figure 7. The configuration ensures a broad field of detection, enabling the robot to identify multiple obstacles simultaneously, thereby aiding in effective obstacle avoidance. In Figure 7, the proximity of obstacles is indicated by red arrows, whereas blue lines denote potential clear paths that the robot can navigate through.

The experimental setup involved varying the robot's start and end points across 15 distinct pairs, with the robot undergoing training for each pair across 5000 episodes. Initial experiments employed a simplistic reward system, penalizing collisions and rewarding goal achievement. However, this initial reward strategy proved inadequate for complex navigation tasks. Consequently, the approach was refined to develop a more sophisticated reward formulation, incorporating multiple factors to better guide the learning process. Following an extensive training regime spanning 75,000 episodes, the algorithm was subjected to a series of 100 evaluation episodes. The outcomes of these evaluations, as presented in Figures 8 and 9, demonstrate a success rate exceeding 50%.

Conclusions and Future Work

In summary, the results of this project underscore the efficacy of the Deep Deterministic Policy Gradient (DDPG) algorithm within simulated environments, marking a significant contribution to the field of reinforcement learning. Despite initial challenges encountered during the preliminary evaluation episodes, DDPG demonstrated a remarkable capacity for recovery, swiftly adapting to achieve an enhanced success rate in subsequent episodes. This resilience highlights DDPG's potential as a robust learning method capable of navigating the complexities of simulated environments effectively.

However, the journey of refining and expanding the capabilities of reinforcement learning algorithms does not end here. One promising direction for future work is the integration of Hindsight Experience Replay (HER). By leveraging the insights gained from previous experiences, HER has the potential to streamline the learning process, particularly in scenarios where reward signals are sparse or difficult to engineer. This approach could significantly mitigate the challenges posed by complex reward structures, enhancing the algorithm's efficiency and effectiveness.

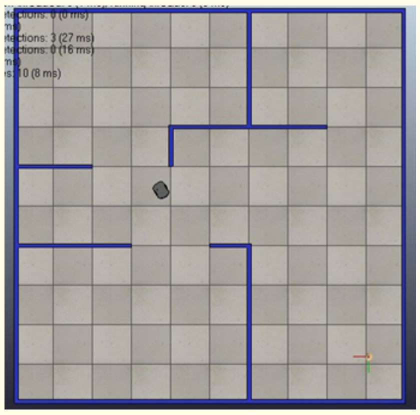

Figure 5: Bot Origin.

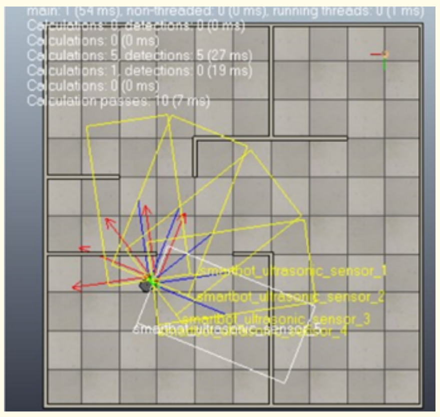
```
main: 1 (54 ms), non-threaded: 0 (0 ms), running threads: 0 (1 ms)
Calculations: 0, detections: 1 (0 ms)
Calculations: 0 (0 ms)
Calculations: 5, detections: 5 (27 ms)
Calculations: 1, detections: 0 (19 ms)
Calculations: 0 (0 ms)
Calculation passes: 10 (7 ms)
```
smartbot_ultrasonic_sensor_1
smartbot_ultrasonic_sensor_2
smartbot_ultrasonic_sensor_3
smartbot_ultrasonic_sensor_4

Figure 6: Gathering Local Information.

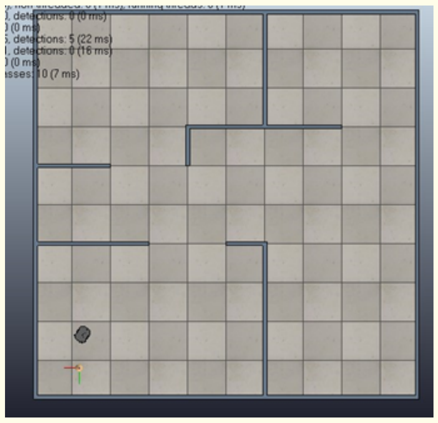

Figure 7: Final Bot Position.

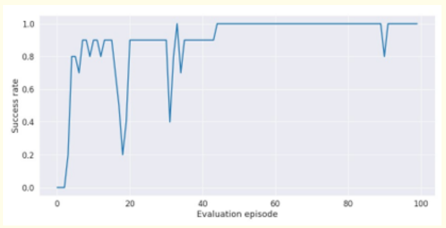

Figure 8: Success Rate.

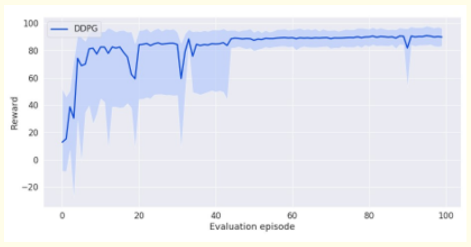

Figure 9: Reward vs Evaluation Episode.

Moreover, the incorporation of vision sensors represents another frontier for exploration. By equipping the algorithm with the ability to perceive and interpret its surroundings through advanced sensory inputs, we can substantially improve its navigational strategies. This enhancement would enable a more nuanced interaction with the environment, facilitating a deeper understanding of the spatial and contextual elements critical for sophisticated decision-making processes.

As advancements are made, these extensions will not only bolster the algorithm's performance but also deepen the understanding of the intricate dynamics at play in reinforcement learning. The integration of HER and vision sensors has the potential to propel the field towards more autonomous, perceptive, and adaptable systems, paving the way for innovations that could transcend simulated environments to impact real-world applications.

In conclusion, this project lays a solid foundation for future research in reinforcement learning, offering valuable insights into the capabilities of DDPG while charting a course for the next generation of algorithms. By embracing these extensions, the boundaries of what is possible can continue to be pushed, driving forward the development of intelligent systems that are more capable, resilient, and attuned to the complexities of their operational environments.

Bibliography

1. G Grisetti., *et al*. "A fully autonomous indoor quadrotor". in: IEEE Transactions on Robotics 28 (2012): 90-100.
2. MW Mueller and R D'Andrea. "A model predictive controller for quadrocopter state interception". In: Control Conference (ECC), 2013 European (2013): 1383-1389.
3. D Silver., *et al*. "Playing atari with deep reinforcement learning". in: arXiv preprint arXiv:1312.5602, 2013. (2013).
4. Richard S Sutton and Shangtong Zhang. "A Deeper Look at Experience Replay". In: NIPS 2017 Deep Reinforcement Learning Symposium (2017).
5. Stephen James., *et al*. "PyRep: Bringing V-REP to Deep Robot Learning". In: arXiv preprint arXiv:1906.11176 (2019).
6. Alexander Pritzel., *et al*. "Continuous control with deep reinforcement learning". In: Google Deepmind (2019).
7. Jiahao Lu., *et al*. "Efficient hindsight reinforcement learning using demonstrations for robotic tasks with sparse rewards". In: International Journal of Advanced Robotic Systems (2020).
8. Ankit Choudhary. "A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python".
9. Or Rivlin. "Reinforcement Learning with Hindsight Experience Replay".
10. Souphis. Open-AI Gym extension for robotics based on V-REP.
11. Wikipedia. Q-learning.